Towards network optimization using Graph Neural Networks

by Paul Almasan

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS: Computer Networks and Distributed Systems

23rd October 2019

SUPERVISOR:

Prof. Alberto Cabellos Aparicio

CO-SUPERVISOR:

Prof. Pere Barlet Ros

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Abstract

Recent advances in Deep Reinforcement Learning have shown a significant improvement in decision-making problems. An example of the successful application is in the use of DRL to design an algorithm to allocate limited resources to different tasks in a computer cluster. Another successful example is DRL in robotics, where a robot can learn a policy to map video images to robot's actions. These tasks, and many others from different fields, have been typically performed by the design of complex heuristics adapted to a simplification of the real problem. These heuristics are not flexible and they have difficulties to adapt to more sophisticated scenarios. DRL, on the other hand, is able to learn from past experiences and approximate its actions to scenarios never seen before. In DRL there is an agent interacting with the environment, and for each action he makes he will receive a positive or negative reward, indicating if it was a good or bad action respectively. Thus, the agent is capable to remember a similar state to the one he is facing and to pick the action that would give him a higher reward.

The networking community has started to investigate how DRL can provide a new breed of solutions to relevant optimization problems. Some of these problems include to find the optimal routing given input traffic demands or network resource management. However, most of the state-of-the-art DRL-based networking techniques fail to generalize, meaning that they can only operate over network topologies seen during training. The reason behind this important limitation is that existing DRL networking solutions use standard neural networks (e.g., fully connected, convolutional neural networks, etc.), which are unable to learn graph-structured information. Computer networks are inherently represented as graphs, and thus, it's an important aspect to learn this graph-structured information for solving network optimization problems.

In this work we study the use of Graph Neural Networks in combination with DRL to solve network optimization problems. GNN have been recently proposed to model graphs and they showed their ability to generalize to other topologies. Having a GNN able to model a graph, we study a novel DRL+GNN architecture, able to learn, operate and generalize over arbitrary network topologies. To showcase its generalization capabilities, we evaluate it on an Optical Transport Network scenario, where the agent needs to allocate traffic demands efficiently. Our results show that our DRL+GNN agent is able to achieve outstanding performance in topologies unseen during training.

The presented implementation considers a scenario where a centralized DRL agent has a complete view of a given network topology. This agent will receive different kind of traffic demands that will have to allocate on the graph. These demands are stochastic and the agent has no information about the future demands. Thus, given a demand, his role is to allocate the resources in the network that would satisfy the demand, trying to maximize the network links utilization in the long term. To assess its performance, we trained the agent on one topology and we evaluated it on another topology not seen during training.

Acknowledgments

To Prof. Albert Cabellos and Pere Barlet for the opportunity of carrying out this work and for the help and support received.

To the research group in UPC that helped with the development of this project.

To Dr. Krzystof Rusek, from AGH University of Science and Technology, for all the technical support.

Finally, I would like to express my gratitude to my family and friends for encouraging throughout my studies and through the process of this thesis.

Contents

1. Introduction

Recent advances in Deep Reinforcement Learning (DRL) showed a significant improvement in decision-making and automated control problems [9][10]. In this context, the network community has adopted DRL as a key technology to provide a new breed of network optimization problems (e.g. routing) with the goal of enabling self-driving networks [5]. However, existing DRL-based solutions still fail to generalize when applied to network-related scenarios. This hampers the ability of the DRL agent to make good decisions when facing a network state not seen during training.

Most of existing DRL proposals for networking can only operate over the same network topology seen during training [11][12] and thus, cannot generalize and efficiently operate over unseen network topologies. The main reason behind this strong limitation is that computer networks are fundamentally represented as graphs. For instance, the network topology and routing policy are typically represented as such. However, state-of-the-art proposals [11][13][14] use traditional neural network (NN) architectures (e.g. fully-connected, Convolutional Neural Networks) that are not well suited to model graph-structured information.

Recently, Graph Neural Network (GNN) [15] have been proposed to model and operate on graphs with the aim of achieving relational reasoning and combinatorial generalization. In other words, GNNs facilitate the learning of relations between entities in a graph and the rules for composing them. GNN have been recently proposed in the field of networking modelling and optimization [16]. In this work, we present a pioneering DRL+GNN architecture for networking. Our novel architecture is able to operate and optimize problems while generalizing to unseen topologies. Specifically, the GNN used by the DRL agent is inspired by Message-passing Neural Networks (MPNN) [17]. Such MPNN-based models are used in chemistry to develop new compounds. With this framework we design a GNN that captures meaningfully the relation between paths and the links on a network topology.

To analyse the generalization capabilities, we experimentally evaluate our proposed DRL+GNN architecture in an Optical Transport Network (OTN) scenario. Specifically, the DRL+GNN agent needs to learn an efficient policy to maximize the number of allocated traffic demands over a topology. Our results show that our agent is able to efficiently operate over a network not seen during training with outstanding performance. At the best of our knowledge this is the first time a DRL agent has been combined with a GNN to address networking problems. We hope that our novel architecture represents a baseline to be adapted to other scenarios.

Objectives

The main focus of this work is to use GNNs to model a network and DRL as an optimization algorithm to find the routing of incoming traffic demands. Therefore, the objectives we defined can be summarized as follows:

- To design and implement a GNN-based model using TensorFlow that is able to model the network topology and generalize to topologies not seen during training. As we work in an OTN environment, the model will have to model the link features which contain relevant information such as link capacity or the bandwidth allocated.
- To design a DRL algorithm that would integrate the GNN model. This algorithm would learn to interact with the network in order to achieve the maximum cumulative reward in the long term.

2. Background

Knowledge Defined Networking

D. Clark et al. proposed "A Knowledge Plane for the Internet" [1] , a new architecture that relies on Machine Learning (ML) and cognitive techniques to operate the network. A Knowledge Plane (KP) would bring many advantages to networking, such as automation and action recommendation, and it has the potential to represent a paradigm shift on the way we operate and optimize data networks.

One of the biggest challenges when applying ML for network operation and control is that networks are inherently distributed systems, where each node (i.e., switch, router) has only a partial view and control over the complete network. To learn from nodes that only have a partial vision of the system can be very complex. There is some effort being done towards the logical centralization control, which will ease the complexity of learning in a distributed environment. Specifically, the Software-Defined Networking (SDN) paradigm [2] decouples control from the data plane, providing a logically centralized control plane (i.e. a single logical point in the network with knowledge of the whole).

In addition to the "softwarization" of the network, current network data plane elements are equipped with improved computing and storage capabilities. This enabled a new breed of network monitoring techniques commonly referred to as network telemetry [4]. These techniques provide real-time packet and flow-granularity information, together with configuration and network state monitoring data, to a centralized Network Analytics (NA) platform. In this context, telemetry and analytics technologies provide a richer view of the network compared to what was possible with conventional network management approaches.

The centralized control offered by SDN, combined with a rich centralized view of the network provided by network analytics, enable the deployment of the KP concept proposed in [1]. In this context, the KP can use various ML approaches, such as Deep Learning (DL) techniques, to gather knowledge about the network, and exploit that knowledge to control the network using logically centralized control capabilities provided by SDN. This new paradigm, resulting from combining SDN, telemetry, Network Analytics, and the Knowledge Plane is known as Knowledge-Defined Networking (KDN) [5]. The work from this thesis is encompassed in the KDN paradigm, where we have a central entity with absolute view and knowledge of the underlying network structure and state.

Graph Neural Networks

Graph Neural Networks (GNNs) is a novel family of neural networks designed to operate over graph-structured information. They were introduced in [15] and numerous variants have been developed since [18][19][20]. In its basic form, they consist in associating some initial states to the different elements in a graph and then combine them considering how these elements are related in the graph. An iterative algorithm updates the state elements and uses the final states to produce an output. The particularities of the problem to solve will determine which GNN variant to use, which elements of the graph (edges or nodes) are considered, etc.

$$M_k^{t+1} = \sum_{i \in N(k)} m(h_k^t, h_i^t) \quad (1)$$

$$h_k^{t+1} = u(h_k^t, M_k^t) \quad (2)$$

Message Passing Neural Networks (MPNN) [17] are a well-known type of GNNs that use an iterative message passing algorithm to propagate information between the nodes of the graph. In a message-passing step, each node *K* receives messages from all the nodes in its neighbourhood, denoted by *N(k)*. Messages are generated by applying a message function *m(·)* to the hidden states of node pairs in the graph, and then are combined by an aggregation function, for instance, a sum (Eq. 1). Finally, an update function *u(·)* is used to compute a new hidden state for every node (Eq. 2).

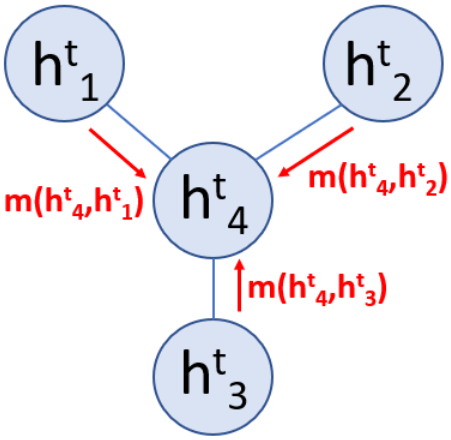

Figure 1

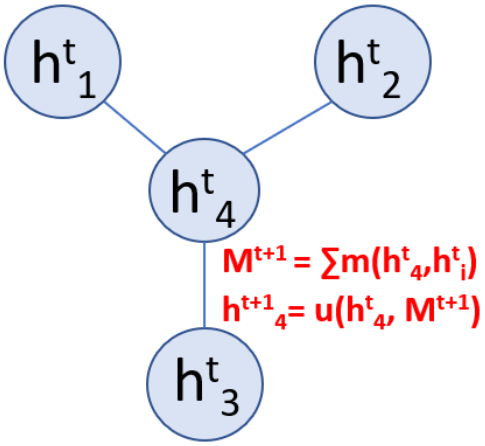

Figure 2

In Figure 1 and Figure 2 we can observe graphically one step of the message passing. At a time step *t*, all nodes have a hidden state vector *h*, indicating the features of that node at this point in time. Each node is identified by the subscript of the hidden state vector h_i where $1 \leq i \leq 4$. In Figure 1 we focus on node 4 at time step t where he receives a message from all his neighbours. The messages correspond to the hidden state vector of his neighbours respectively. Afterwards, node 4 aggregates the messages received from his neighbours (see Figure 2). The aggregation operation is typically a sum, but it can be changed by other methods. Finally, node 4 will update its status using his current state h^t_i and the aggregation of all messages received from its neighbours. This same procedure is repeated at the same time for all nodes in the graph until convergence.

Functions *m(·)* and *u(·)* are differentiable functions, and consequently may be learned by neural networks. After a certain number of iterations, the final node states are used by a readout function *r(·)* to produce an output for the given task. This function can be also implemented by a neural network and is typically tasked to predict properties of individual nodes (e.g., the node's class) or global properties of the graph.

RouteNet

GNNs have been able to achieve relevant performance results in multiple domains where data is typically structured as a graph [17]. Since computer networks are fundamentally represented as graphs, it is inherent in their design that GNNs offer unique advantages for network modelling compared to traditional neural network architectures (e.g., fully connected NN, convolutional NN). An example of this is RouteNet, a novel neural network architecture which has shown its potential for network performance prediction [16], even when making predictions on network topologies not seen during training [21]. RouteNet is a GNN-based model able to understand the complex relationship between topology, routing and input traffic to produce accurate estimates of the per-source/destination pair mean delay and jitter. This is achieved by modelling the relationships of the links in topologies with the source-destination paths resulting from the routing schemes and the traffic flowing through them.

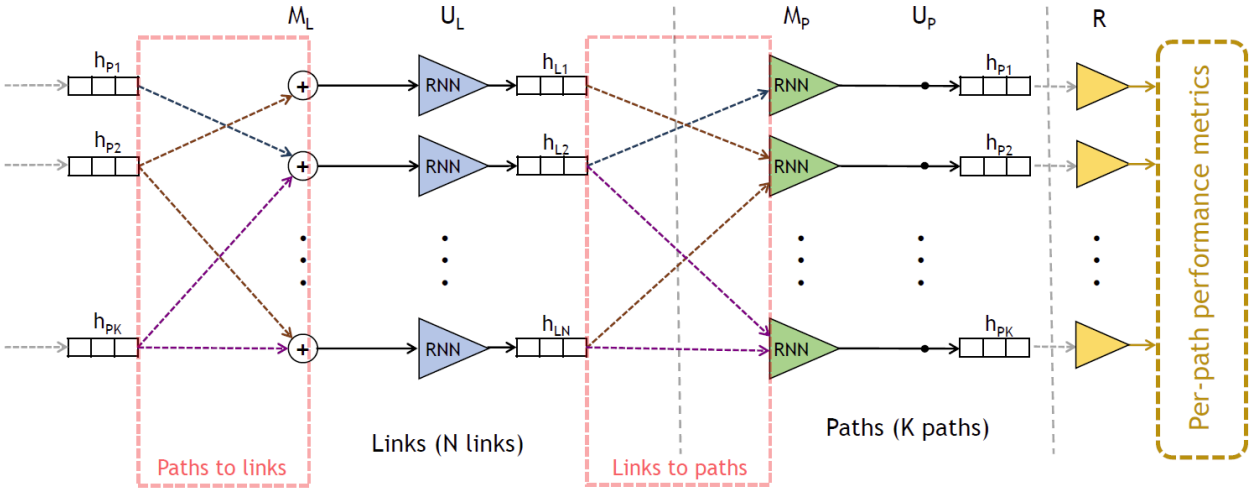

Figure 3

In Figure 3 we can see a graphical interpretation of the message passing performed in RouteNet. The main entities are the links and the paths, obtained from combining a traffic matrix, a routing and the network topology. The traffic matrix indicates for each source-destination how many bits per second are sent. There are two steps in the RouteNet message passing: paths to links and links to paths. In the first, each path sends a message to all the links from the paths. Then, for each link the information is aggregated using a summation. This information is passed through a Recurrent NN (RNN), responsible for learning the sequence of paths passing through each link. The second step, each link sends a message to all the paths passing by that link. The paths hidden states are updated using the output of an RNN, which is responsible for learning from the sequence of links forming each path.

Deep Reinforcement Learning

DRL algorithms aims at learning a strategy that leads to maximize the cumulative reward in an optimization problem. DRL agents start from tabula rasa. This means that they have no previous expert knowledge about the environment where they operate. They have only a set of possible actions and learn the optimal strategy after an iterative process that explores the action and observation spaces. The learning process consists in a set of actions A and a set of states S. Given a state $s \in S$, the agent will perform an action $a \in A$ that produces a transition to a new state $s' \in S$, and will provide the agent with a reward r . This optimization problem can be modelled as a Markov Decision Process (MDP). However, finding the solution of the MDP requires to evaluate all the possible combinations of state-action pairs.

An alternative to solve the MDP is using Reinforcement Learning (RL). Q-learning [22] is a value-based RL algorithm whose goal is to make an agent learn a policy $\pi : S \rightarrow A$. The algorithm creates a table (a.k.a. q-table) with all the possible combinations of states and actions. At the beginning of the training, the table is initialized (e.g., zeros or random values) and during training the agent updates these values according to the rewards received after selecting an action. These values, called q-values, represent the expected reward assuming the agent is in a state s and performs action a. During training, q-values are updated using the Bellman equation (see Eq. 3) where $r(s,a)$ is the reward obtained from selecting action a in state s , $Q(s',a)$ is the q-value function and $\gamma \in [0 , 1]$ is the discount factor, which represents the importance of the rewards obtained in the future.

$$Q(s, a) = r(s, a) + \gamma \max_{a} Q(s', a) \qquad (3)$$

Deep Q-Network (DQN) [9] is a more advanced algorithm based on Q-learning that uses a deep NN to approximate the q-value function. As the q-table size becomes larger, Q-learning has difficulties to learn a policy from high dimensional state and action spaces. To overcome this problem, they proposed to use a deep NN as q-value function estimator. This method uses an experience replay buffer to store past sequential experiences (i.e. stores tuples of $\{s,a,r,s'\}$). While training the neural network, the temporal correlation is broken by sampling randomly from the experience buffer.

Policy Gradients (PG) methods is a family of algorithms that instead of learning a value function, they try to optimize the policy function π. Nevertheless, these methods suffer from high variance and low convergence. This is because the $log(.)$ operation used to update the policy gradient introduces high variability. This high variability will make noisy gradient and will cause unstable learning. More advanced algorithms that combine value-based and policy gradients were proposed to leverage the benefits of both techniques. These algorithms are called Actor-Critic [30] and they combine an actor agent and a critic agent. The actor outputs the best action to make and the critic evaluates the action using a value function.

A DRL-based solution must define two main elements: (i) the observation space and (ii) the action space. The observation space describes the state of the environment. The action space describes the set of actions the DRL agent can make to modify the environment. Typically, the network state is represented as a matrix containing the utilization or the traffic aggregated on links or nodes [11]. We argue that this approach is not appropriate as a matrix does not represent well graph-structured information. For example, two graphs that differ in one link will have a similar matrix but from a networking point of view both networks can have very different performance. In this work, the network state is going to be represented by a graph with features on the edges. As a proof-of-concept, we are going to use the DQN algorithm because of its simplicity and its successful applications in other fields [9].

State of the art

To find the optimal routing given the traffic between each source-destination pair in a network topology is a fundamental problem in networking. In the last years, different solutions have been proposed to this problem. Boyan & Littman [7] propose Q-routing, a distributed packet routing algorithm where they used a table-based representation of the routing policy. In this table, each node *x* has a value which indicates the estimated time it takes to deliver a packet bound for a node *d* using the current node's neighbour *y*. Because of the fixed size of the table, the solution does not generalize to other topologies. Chen et al. [11] propose a RL solution based on Q-learning where they use a convolutional NN to extract graph features. Particularly, they represent a network state with the links utilization using binary arrays to indicate if the number of frequency slot is available (1) or occupied (0). Similar to Q-routing, the architecture presented has a fixed input size, forbidding the model to generalize to different network topologies of variable size. Mao et al. [33] propose to represent the network state using a matrix with the traffic demand aggregated in every router. Both solutions [11][33] define a discrete action space for the agent where an action is to select a path among a number of candidate paths. One drawback of this representation is that the DRL agent must abstract knowledge from the link-level features to the path-level represented in the action space. You et al. [6] propose a distributed Multi Agent Reinforcement Learning (MARL) solution. They extend the Q-routing solution to a multi agent environment where each agent has a partial view of the network and they take actions that lead to the minimization of the average delivery time. Because of the inherent distributed nature of MARL, the solution seems to be scalable but the problem complexity increases exponentially to the number of nodes added to the topology. Suarez et al. [8] propose a DRL-based solution in OTN where they design an elaborated representation of the network state that helps the agent to capture easily the singularities of the network topologies. Their network state representation reduces the level of knowledge abstraction required for the DRL agent, allowing the agent to learn easily and faster.

3. Network optimization scenario

Finding the optimal routing configuration for a set of traffic demands is a NP-hard problem [32]. This makes it necessary to explore alternative solutions (e.g., heuristics) with the aim of maximizing the performance at an affordable cost. However, hand-crafted heuristics perform well for specific scenarios but they lack the generalization capabilities to other scenarios. In networking, this is translated to the generalization capabilities over different networks. In other words, to find a routing configuration using some heuristic might work for a given topology, but it will perform poorly if we run the same algorithm on a larger topology (e.g. adding more nodes and links to the original topology). In this work, we explore the potential of a GNN-based DRL agent to operate and generalize over routing scenarios involving diverse network topologies. As a first approach, we consider a routing scenario in Optical Transport Networks (OTN). In this scenario, the DRL agent needs to make routing decisions on every traffic demand as it comes.

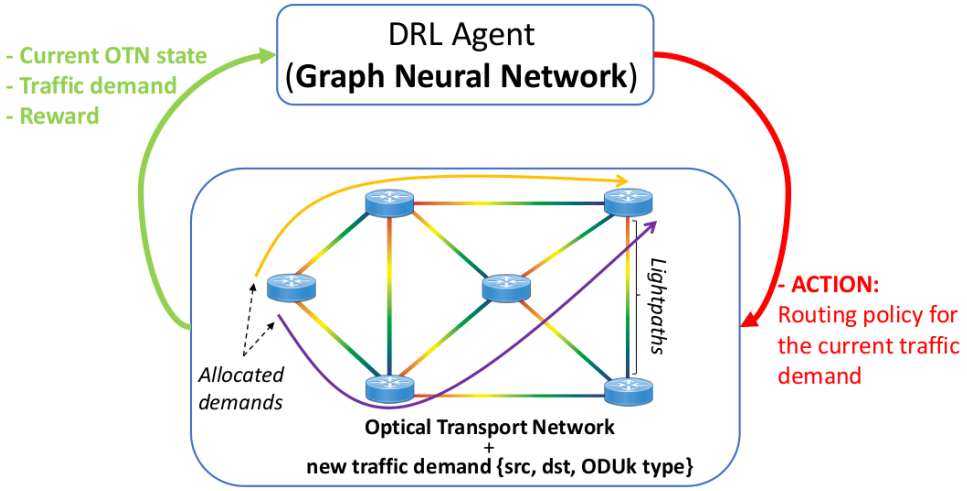

Figure 4

The DRL agent operates at the level of the electrical domain, over a logical topology where the nodes represent Reconfigurable Optical Add-Drop Multiplexer (ROADM) nodes and edges some predefined lightpaths connecting them (see Figure 4). Thus, the DRL agent receives a traffic demand defined by the tuple *{src,dst,bandwidth}*, and is tasked to route the traffic demand through a particular sequence of lightpaths (i.e. an end-to-end path) that connect the source and the destination of such demand. Traffic demands are considered requests of Optical Data Units (ODU) whose bandwidth demands are defined in the ITU-T Recommendation G.709 [23]. Once the demand is allocated, the links available bandwidth is updated and the agent receives a new demand tuple. This process is repeated until there is a demand that cannot be allocated (i.e. the edges have no more free capacity to allocate the new demand). For each allocated traffic demand, the agent receives a positive reward *r=1* if it was allocated successfully or *r=0* if after the allocation there is some link that surpassed its capacity.

In our scenario, the routing problem is defined as finding the optimal strategy to route incoming source-destination traffic demands with the goal of saving network resources in the long-term. In other words, to find the strategy that will allow to allocate more demands. We consider that a demand is properly allocated if there is enough available capacity in all the lightpaths forming the end-to-end path selected. The demands do not expire, occupying the lightpaths until the end of an episode. This implies a challenging task for the DRL agent, since it has not only to identify critical resources on networks (e.g. potential bottlenecks) but also has to deal with the uncertainty in the generation of future traffic demands.

One of the reasons behind choosing this evaluation scenario is that it's a classical problem for OTN, for which a close-to-optimal heuristic is well known. The heuristic we are going to compare our solution with is the Shortest Available Path (SAP), which consists on allocating the demands on one of the *k=4* shortest available paths for each source-destination node pair. This will serve as a baseline benchmark to validate the generalization performance of our model. As a lower bound, we use a random available path policy, meaning that the agent allocates the resources on one of the *k=4* shortest available paths, trying to have the loads balanced. This policy can also be seen as a load balancing as the resources are allocated on the candidate paths following a uniform distribution. We picked the value *k=4* in all the experiments to maintain a reduced number of paths [8]. Having a large k value implies a larger state dimensionality, which means larger processing cost and a more complex learning process, as the actions space to explore is larger.

4. GNN-Based DRL Agent design

In this section we describe the DRL+GNN agent proposed in this work. Our agent implements the DQN algorithm [24], where the q-value function is modelled by a GNN. Algorithm 1 represents the pseudo-code including the training process. The agent has a main loop that is repeated over a total number of iterations, and two nested loops that represent periodic evaluation and training periods. The variables *{s, d, r}* represent the network state, the current traffic demand to allocate and the obtained reward respectively. The training loop (lines 2~12) is executed over *Training_eps* episodes. For each new demand, the agent selects a discrete action that corresponds to an end-to-end path. After every decision, it receives a reward that is stored into an experience replay buffer together with the state and the action. Then, function *agt.replay()* takes randomly some samples from this buffer to train the GNN model by exploiting the input graph-structured information. The evaluation loop (lines 13~20) is executed during *Evaluation_eps* episodes, where one episode starts with an empty network and finishes when a demand is allocated into a path that do not have enough available capacity for the current traffic demand. Particularly, this occurs when at least one lightpath in the end-to-end path selected does not have enough available capacity. After the evaluation loop is completed, we store the mean reward accumulated over all the evaluation episodes. This is used to represent the evolution of the performance achieved by the DRL agent during the training,

Algorithm 1 DRL Agent Training algorithm

```
1:  for it in Iterations do
2:      for episode in Training_eps do
3:          s, d, src, dst ← env.reset_env()
4:          reward ← 0
5:          while TRUE do
6:              a, s' ← agt.act(s, d, src, dst)
7:              r, done, d', src', dst' ← env.step(s')
8:              agt.rmb(s, d, src, dst, a, r, s', d', src', dst')
9:              reward ← reward + r
10:             If done == TRUE : break
11:             If len(agt.mem) > batch_size : agt.replay()
12:             d ← d', s ← s', dst ← dst'
13:     for episode in Evaluation_eps do
14:         s, d, src, dst ← env.reset_env()
15:         reward ← 0
16:         while TRUE do
17:             a, s' ← agt.act(s, d, src, dst)
18:             r, done, d', src', dst' ← env.step(s')
19:             reward ← reward + r
20:             If done then break
```

The GNN model is designed following a MPNN fashion using the link entity and we perform the message passing step between all links. Each link has a feature vector (a.k.a. hidden state) were all the relevant features to solve our problem are going to be stored. For each edge, we store in the hidden state the betweenness centrality of each edge. This centrality measurement is computed in the following way: for every pair of source-destination nodes, we compute *k=4* shortest paths and we store, in each link that these paths are traversing, a counter indicating how many shortest paths go through that link. Another feature we store in the hidden state is the edge capacity. All the capacities are initially set to 200 and these are going to be reduced if the agent allocates resources on the links of a given path. Finally, contrary to many DQN algorithms, we indicate the action performed by the agent in the graph. In our specific use-case, given the node pair source-destination from a traffic demand, the possible actions are *a=1...4*, where *a=1* means to allocate on the 1st shortest path and *a=4* means to allocate on the 4th shortest path. Thus, we indicate this action by adding to the links from the allocated path the demand value the agent is allocating. It's important to remark that each value added to the hidden state is normalized by subtracting the mean and dividing by the standard deviation. This is done to facilitate the learning process and stabilize the learning of the neural networks.

Once the corresponding features are introduced in the affected links after allocating a path, the GNN performs the message passing. Thus, for each link *L* the message function from Eq.1 will take as input the hidden state of the link h_L and the hidden state of a neighbour link h_i *where* $i \in N(i)$. The same process is repeated for all neighbours of link *L* and for all links from the graph. After iterating over all links, the outputs are aggregated using an element-wise sum, resulting on a new feature vector for each link. The resulting vectors are combined with the respective previous link hidden state in a Recurrent NN (RNN). The output of the RNN is going to be the new link hidden state and the process starts again. This whole process is repeated *T* times until convergence. Finally, the outputs of the RNN are aggregated using a sum, passing the result as input to a fully connected neural network (a.k.a. readout). This will output a single value, indicating the q-value of the allocated demand. This value is going to indicate how likely is that the action performed by the agent would return a high reward. In Figure 5 we can observe a graphical representation of the message passing algorithm implemented in this work, where M represents a one-layer neural network, RNN is a GRU recurrent neural network and Readout is a 2-layer fully connected neural network.

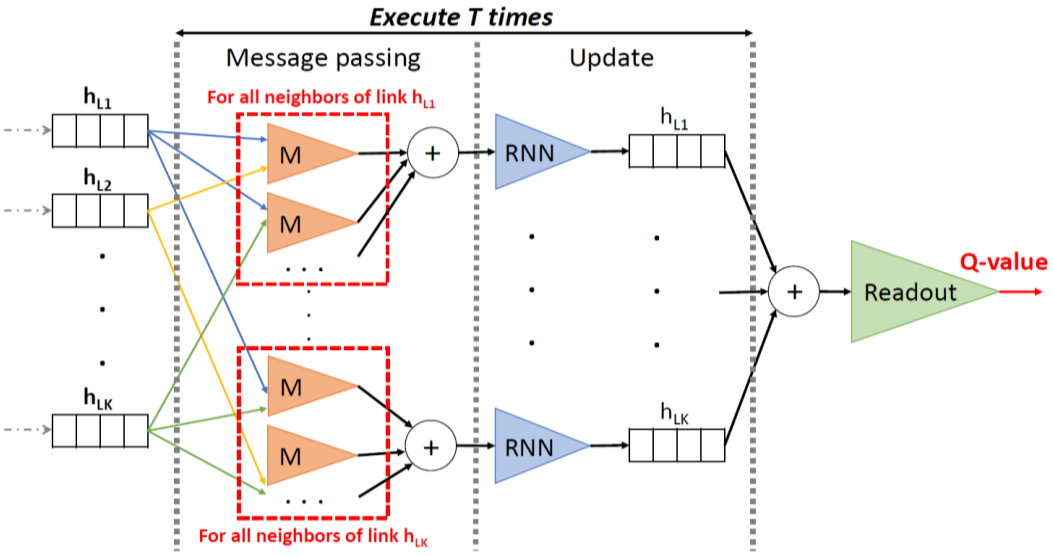

Figure 5

To simulate the environment, we implemented the *env.step()* and *env.reset_env()* functions in the Gym framework [25]. The *env.step()* simulates the transition of the network state after a traffic demand is allocated and generates a new traffic demand for the following step. In case there is no enough free capacity in the path selected, the agent is notified with a flag (done). The *env.reset_env()* method is executed after every episode end and is responsible for resetting the environment and generate the first traffic request for the next episode. Also, the agent uses the *agt.rmb()* function to store transitions in the experience replay buffer. For this buffer, we implemented a cyclic replay memory (i.e. once the memory is full, oldest experiences are removed) to avoid that the agent is trained on very old samples as the training evolves. The *agt.act()* method selects one over a set of possible actions given an input state following an ε-*greedy* strategy [24] during training. This strategy is used to indicate the agent if to pick the actions totally random (ε=1.0) or pick the one with the highest q-value (ε=0.01).

In Code 1 we can see the original code of the message passing implemented in TensorFlow. In lines 2~12 we are reading the input data samples and parsing the tensors using the variable *num_edges* stored for each sample. This is done to enable the message passing be able to process samples from graphs of different sizes, which implies tensors of different lengths. In lines 13~14 we read the identifiers of the encoded links from the graph (first) with all their neighbors (second) respectively. These identifiers are used to read the hidden states of all the links and combine them to perform the forward pass through the *Message (M)* neural network. Afterwards, for every link from the graph we aggregate the results, by using a segment sum, obtained from *M*. The resulting vectors are passed to an RNN in the *Update* function from line 22. After repeating this process for *T* times, the resulting hidden states of the links are aggregated in a sum resulting in a unique vector. Finally, this vector is processed by the Readout neural network in line 27, which will output a single q-value.

```
def call(self, inputs, training=False):
        capacities=tf.reshape(inputs['capacities'][0:inputs['num_edges']],
        [inputs['num_edges'],1])
        betweenness=tf.reshape(inputs['betweenness'][0:inputs['num_edges']],
        [inputs['num_edges'],1])
        bw_allocated=tf.reshape(inputs['bw_allocated'][0:inputs['num_edges']
        ], [inputs['num_edges'], 1])

        hiddenStates=tf.concat([capacities,betweenness,bw_allocated],
        axis=1)
        paddings=tf.constant([[0, 0],[0, self.hparams['link_state_dim']-3]])
        link_state = tf.pad(hiddenStates, paddings, "CONSTANT")
        first = inputs['first'][0:inputs['length']]
        second = inputs['second'][0:inputs['length']]
        for _ in range(self.hparams['T']):
            mainNodes = tf.gather(link_state, first)
            neighNodes = tf.gather(link_state, second)
            nodesConcat=tf.concat([mainNodes,neighNodes],axis=1)
            outputs=self.Message(nodesConcat)
            nodes_inputs = tf.math.unsorted_segment_sum  (data=outputs,
            segment_ids=second,num_segments=inputs['num_edges'])
            outputs,links_state_list=self.Update(nodes_inputs,[link_state])

            link_state = links_state_list[0]

        nodes_combi_outputs = tf.math.reduce_sum(link_state, axis=0)
        r = self.Readout([nodes_combi_outputs],training=training)
        return r
```

Code 1

5. Experimental Results

In this section we train and evaluate our GNN-based DRL agent to efficiently allocate traffic demands in an OTN routing scenario. We set the hyperparameters and the optimizer that best fits in this problem by running different experiments. The main goal of these experiments is to see if the implemented GNN model is able to learn the graph structure information and interact with the network by allocating "optimally" the incoming traffic requests. Moreover, we also want to test the generalization capabilities of GNNS.

In all the experiments, we used the National Science Foundation Network (NSFNET) [28] and the German Backbone Network (GBN) [29] network topologies. In Figure 7 and Figure 6 we can see the representation of the GBN and NSFNET topologies respectively. To test the generalization capabilities of GNNs, we used NSFNET during the training process and GBN on evaluation only. This means that the DRL agent never saw the GBN topology during training.

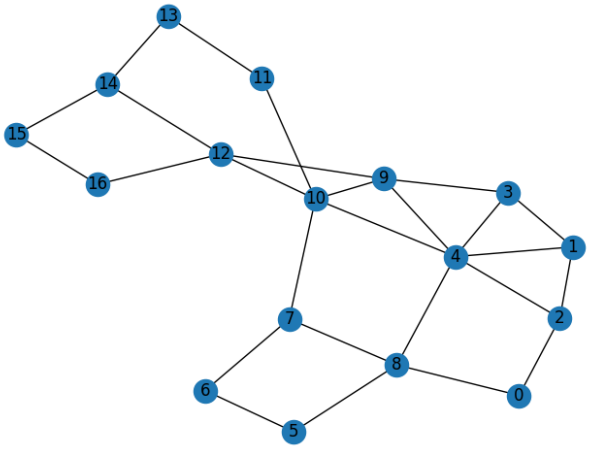

Figure 7 : GBN

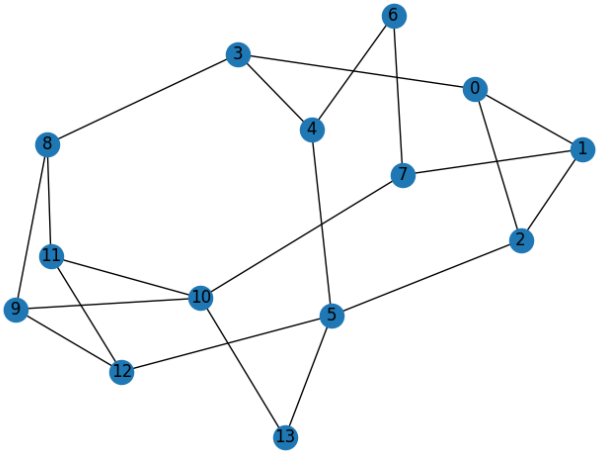

Figure 6 : NSFNET

Evaluation setup

We implemented the DRL environment using the OpenAI Gym framework. For the sake of simplicity, we consider 2 types of traffic demands (ODU3, ODU4) whose bandwidth requirements are expressed in terms of multiples of ODU0 signals (32 and 64 ODU0 signals respectively). When the DRL agent allocates a demand, it receives an immediate reward being the bandwidth (in ODU bandwidth units) of the current traffic demand if it was properly allocated, otherwise the reward is 0. Traffic demands are generated on every step by randomly selecting a source-destination pair in the network and an ODUk type demand that represents the bandwidth.

Preliminary experiments were carried to choose an appropriate optimizer and hyperparameter values for our DRL agent. We compare three well-known optimizers (i.e. Adam, RMSProp and Stochastic Gradient Descent with optimizations) to choose the one that has a more stable learning process and converges faster to a solution. In Figure 8 we can observe the results of the comparison and clearly Stochastic Gradient Descent [26] method with Nesterov Momentum [27] offers the fastest convergence and higher test score (SGDOptimized). Regarding the hyperparameters, we use a learning rate of 10−4, and a momentum of 0.9. Each training episode consists on a 100 independent sequences of traffic demands where each sequence finishes when a demand cannot be allocated on the network. After each training episode, we evaluate the model on the same topology used in the training loop. This process consists of 50 evaluation episodes and we store the average of them. For the ε-greedy exploration strategy, we start with ε =1.0 that is maintained during 1000 episodes. Then, ε decays exponentially every 2 training episodes. For the experience replay buffer, we select a size of 5000 samples.

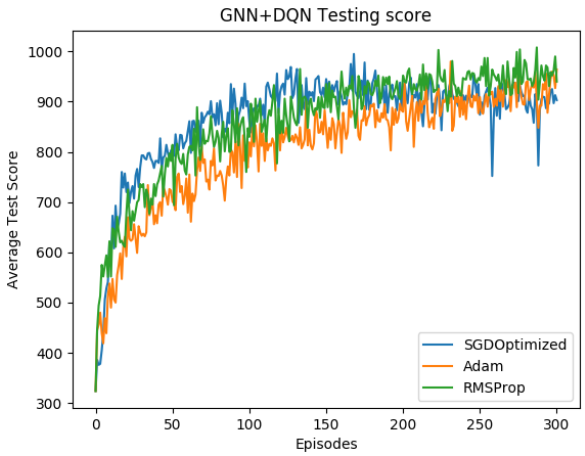

Figure 8

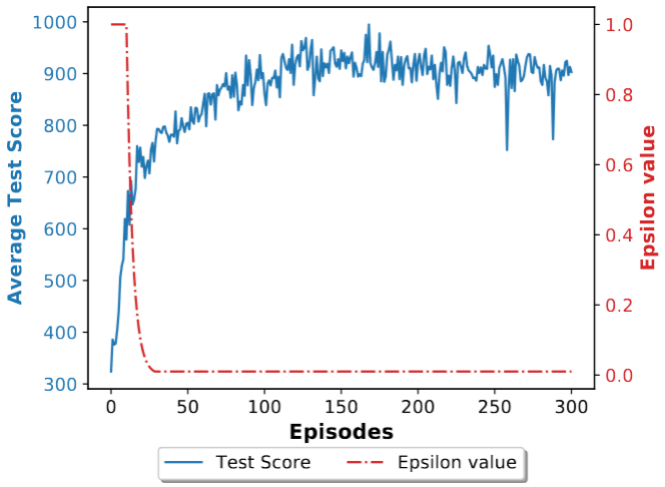

Figure 9

Training

We train the DRL agent on an OTN routing scenario with the 14-node NSFNET topology, where we consider that the edges represent lightpaths with capacity for 200 ODU0 signals. During training, the agent receives traffic demands and allocates them on one of the k=4 shortest paths available in the action set. We run 100 episodes, store the output in the experience replay buffer and train the GNN by selecting randomly one sample from the buffer. For the evaluation, we run 50 episodes and compute the average cumulative reward obtained over all of them.

In Figure 9 we show the evaluation score of the GNN-based model during training. The score is computed by evaluating the agent on the NSFNET topology, which is the only one the agent sees during the training process. We also show the evolution of ε during the training. As we can observe, when epsilon starts to decay (i.e., around episode 15) there is a stable increase in the score achieved by the agent. This suggests that, at this point, the GNN is already in a positive phase of training and it is possible to use its q-value estimates to make a smarter exploration of the action space.

Generalization over other network scenarios

To evaluate the generalization capability of our agent, we select the version of the agent with highest score during the training and evaluate it on a scenario of the 17-node GBN topology. Note that the agent has not seen any sample from this topology during training. In order to benchmark its performance, we compare it against the "Shortest Available Path" (SAP) policy. This routing policy typically represents a performance close to the optimal MDP solution in our OTN routing scenario. To have a lower bound, we compare with the “Random Available Path” (RAND) policy. This policy picks randomly one path among the shortest available paths where the agent will allocate the incoming traffic demand.

Figure 11 shows the performance of our GNN-based DRL agent on NSFNET topology against the SAP heuristic and the RAND policy. We can observe that our agent was able to learn a policy that would match the performance of SAP, a heuristic close to the optimal solution, in the same topology used during training. The y-axis represents the score achieved over 50 evaluation episodes (x-axis). The horizontal lines indicate the average score obtained over all the episodes by each strategy. In Figure 10 we can observe the performance of the same agent evaluated on the GBN topology. This plot reveals the ability of our DRL agent to maintain a good performance even when it operates in a routing scenario with a different topology not seen during training. One of the explanations behind the difference in performance between SAP and our DRL agent is that GBN might have critical links (i.e. many shortest paths cross those links) that SAP congests very soon. Thus, a smarter strategy, such as the learned by our agent, would learn that there are some links that will run out of capacity soon, impeding the agent to allocate more demands. Therefore, the agent would look for different ways to allocate the incoming demands and avoid allocating on critical links.

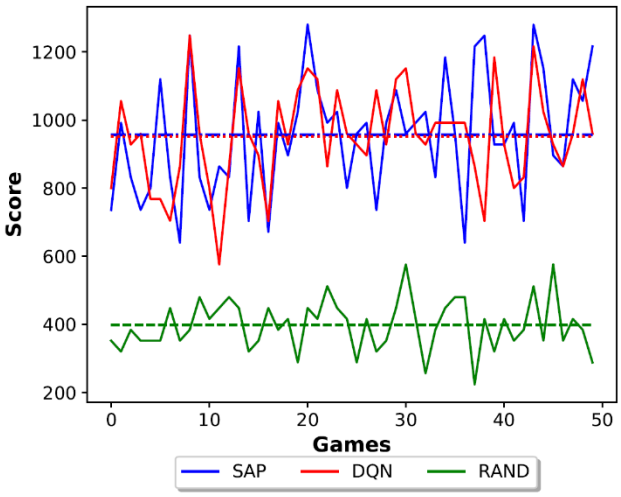

Figure 11

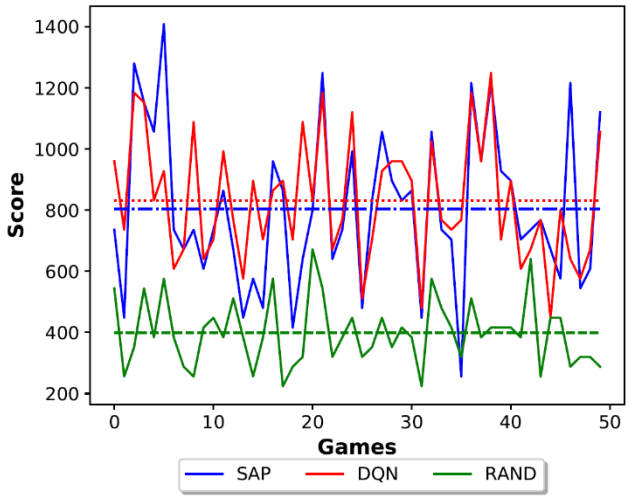

Figure 10

6. Conclusion

In this work, we presented a DRL architecture based on GNNs that is able to generalize to unseen network topologies. The use of GNNs to model the network environment allow the DRL agent to operate in different networks than those used for training. We believe that the lack of generalization was the main obstacle preventing the deployment of existing DRL-based solutions in production networks.

In order to show the generalization capabilities of our DRL+GNN architecture, we selected a classical problem in the field of OTN, for which a close-to-optimal heuristic is well known. This served as a baseline benchmark to validate the generalization performance of our model. Our results show that our model is able to sustain a similar accuracy in a network never seen during training. Previous DRL solutions based on traditional neural network architectures are not able to generalize to other topologies.

Our ongoing work is focused on applying our DRL+GNN architecture to more complex routing and networking problems. Given the generalization performance shown by GNNs for modelling more complex scenarios, we are confident similar results will also be obtained when combined with DRL.

Future Work

The present work is a partial result from an ongoing project with the goal of routing packets efficiently. This goal is challenging because it has been proven that routing is an NP-hard problem. Nevertheless, RL opens many possibilities that would allow as to approximate a close-to-optimal solution. To reach this goal, many steps need to be done:

- The results showed in this work are very promising but the environment is not challenging enough. Therefore, an obvious next step would be to add more different kind of demands to the OTN environment and test the generalization capabilities of the GNN-based DRL agent.
- Another interesting routing scenario would be one where the allocated traffic demands they have a limited amount of time. This means that after they are allocated, they spend some time and afterwards they free the resources they were using. This environment is more realistic and challenging for a GNN as it's changing dynamically.
- More sophisticated DRL techniques could be applied to our problem. To train our DQN algorithm and see some results was a matter of days. Thus, to implement advanced DRL algorithms would allow us to achieve better results and in a considerably less amount of time.

References

[1] Clark, D. D., Partridge, C., Ramming, J. C., & Wroclawski, J. T. (2003, August). A knowledge plane for the internet. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (pp. 3-10). ACM.

[2] Kreutz, D., Ramos, F., Verissimo, P., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. arXiv preprint arXiv:1406.0440.

[3] 2019. ITU-T Recommendation G.709/Y.1331: Interface for the optical transport network. https://www.itu.int/rec/T-REC-G.709/.

[4] Kim, C., Sivaraman, A., Katta, N., Bas, A., Dixit, A., & Wobker, L. J. (2015, August). In-band network telemetry via programmable dataplanes. In ACM SIGCOMM.

[5] Mestres, A., Rodriguez-Natal, A., Carner, J., Barlet-Ros, P., Alarcón, E., Solé, M., ... & Estrada, G. (2017). Knowledge-defined networking. ACM SIGCOMM Computer Communication Review, 47(3), 2-10.

[6] You, X., Li, X., Xu, Y., Feng, H., & Zhao, J. (2019). Toward Packet Routing with Fully-distributed Multi-agent Deep Reinforcement Learning. arXiv preprint arXiv:1905.03494.

[7] Boyan, J. A., & Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In Advances in neural information processing systems (pp. 671-678).

[8] Suárez-Varela, J., Mestres, A., Yu, J., Kuang, L., Feng, H., Cabellos-Aparicio, A., & Barlet-Ros, P. (2019). Routing in optical transport networks with deep reinforcement learning. Journal of Optical Communications and Networking, 11(11), 547-558.

[9] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529.

[10] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Lillicrap, T. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815.

[11] Chen, X., Guo, J., Zhu, Z., Proietti, R., Castro, A., & Yoo, S. J. B. (2018, March). Deep-RMSA: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks. In 2018 Optical Fiber Communications Conference and Exposition (OFC) (pp. 1-3). IEEE.

[12] Lin, S. C., Akyildiz, I. F., Wang, P., & Luo, M. (2016, June). QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach. In 2016 IEEE International Conference on Services Computing (SCC) (pp. 25-33). IEEE.

[13] Mestres, A., Alarcón, E., Ji, Y., & Cabellos-Aparicio, A. (2018, August). Understanding the modeling of computer network delays using neural networks. In Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (pp. 46-52). ACM.

[14] Suárez-Varela, J., Mestres, A., Yu, J., Kuang, L., Feng, H., Barlet-Ros, P., & Cabellos-Aparicio, A. (2019, March). Routing based on deep reinforcement learning in optical transport networks. In Optical Fiber Communication Conference (pp. M2A-6). Optical Society of America.

[15] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. IEEE Transactions on Neural Networks, 20(1), 61-80.

[16] Rusek, K., Suárez-Varela, J., Mestres, A., Barlet-Ros, P., & Cabellos-Aparicio, A. (2019, April). Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN. In Proceedings of the 2019 ACM Symposium on SDN Research (pp. 140-151). ACM.

[17]Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017, August). Neural message passing for quantum chemistry. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 1263-1272). JMLR. org.

[18]Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in neural information processing systems (pp. 3844-3852).

[19]Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2015). Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493.

[20]Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, February). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-1147).

[21]Suárez-Varela, J., Carol-Bosch, S., Rusek, K., Almasan, P., Arias, M., Barlet-Ros, P., & Cabellos-Aparicio, A. (2019, August). Challenging the generalization capabilities of Graph Neural Networks for network modeling. In Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos (pp. 114-115). ACM.

[22]Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

[23]2019. ITU-T Recommendation G.709/Y.1331: Interface for the optical transport network. https://www.itu.int/rec/T-REC-G.709/.

[24]Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[25]Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. arXiv preprint arXiv:1606.01540.

[26]Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010 (pp. 177-186). Physica-Verlag HD.

[27]Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, February). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-1147).

[28]Hei, X., Zhang, J., Bensaou, B., & Cheung, C. C. (2004). Wavelength converter placement in least-load-routing-based optical networks using genetic algorithms. Journal of Optical Networking, 3(5), 363-378.

[29]Pedro, J., Santos, J., & Pires, J. (2011, June). Performance evaluation of integrated OTN/DWDM networks with single-stage multiplexing of optical channel data units. In 2011 13th International Conference on Transparent Optical Networks (pp. 1-4). IEEE.

[30]Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. In Advances in neural information processing systems (pp. 1008-1014).

[31]Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In International conference on machine learning (pp. 1889-1897).

[32]Di Ianni, M. (1998). Efficient delay routing. Theoretical Computer Science, 196(1-2), 131-151.

[33]Mao, B., Fadlullah, Z. M., Tang, F., Kato, N., Akashi, O., Inoue, T., & Mizutani, K. (2017). Routing or computing? The paradigm shift towards intelligent computer network packet transmission based on deep learning. IEEE Transactions on Computers, 66(11), 1946-1960.